PETIT GUIDE DE SIMPLIFICATION EN MAPLE

Version légèrement mise à jour d'un chapitre de l'ouvrage (désormais indisponible), Calcul formel : mode d'emploi *de Claude Gomez, Bruno Salvy, Paul Zimmermann. Masson (1995). Certains commentaires sont devenus obsolètes dans les versions récentes de Maple.*

La résolution de problèmes concrets passe par l'emploi de nombreuses fonctions : `solve`, `subs`, `normal`, `simplify`, `eval`, `fsolve`, `plot`,.... Outre la maîtrise de l'aide en ligne, il faut apprendre à raisonner en termes de classes d'expressions. Chaque fonction Maple s'applique à (et produit) une classe bien définie d'expressions. Reconnaître qu'une expression appartient à telle ou telle classe permet du même coup de savoir quelles fonctions on peut lui appliquer.

Un problème pour lequel cette reconnaissance est essentielle est celui de la simplification d'expressions. C'est autour de ce problème que sont définies les principales classes d'expressions en Maple. En effet, dès qu'il est possible de déterminer si une expression appartenant à une classe est nulle ou non, il est possible d'effectuer des divisions dans cette classe. Autrement, tous les calculs qui demandent une division deviennent hasardeux. Dans les classes les plus simples, il existe une *forme normale*. Sous cette forme, deux expressions représentent le même objet mathématique si et seulement si elles sont identiques. Cependant, la représentation idéale n'est pas toujours la forme normale. Dans le cas des polynômes par exemple, la représentation développée est une forme normale, mais la représentation factorisée permet des calculs de pgcd bien plus rapides. Ce genre d'exemple amène Maple à un compromis. Un certain nombre de simplifications basiques, comme la réduction des rationnels ou la multiplication par zéro, sont effectuées automatiquement ; les autres récritures sont laissées à l'initiative de l'utilisateur auquel des commandes spécialisées sont proposées.

Les principales fonctions permettant de récrire des expressions sont `normal`, `expand`, `combine`, `collect` et `simplify`. Pour bien utiliser ces fonctions, il faut savoir quel type de transformations elles effectuent et à quelle classe d'expressions ces transformations s'appliquent. Ainsi, l'usage aveugle de la fonction `simplify` était très dangereux dans le passé et pouvait conduire à des résultats faux. La situation s'est beaucoup améliorée dans les versions récentes. Un second argument de `simplify` permet par ailleurs de préciser la simplification à effectuer.

Dans ce petit guide, nous allons passer en revue les principales classes d'expressions communes en Maple et les fonctions de manipulation correspondantes. Nous insisterons sur les fonctions de récriture, c'est-à-dire celles qui modifient la forme d'une expression sans changer sa signification mathématique. Un premier aperçu est donné par le tableau 1.

Date: Version du 28 février 2008.

1. Classes élémentaires

Les classes élémentaires sont formées d'expressions sans variable, c'est-à-dire de constantes : entiers, rationnels, nombres flottants, booléens, résidus modulo p et nombres p-adiques.

1.1. **Entiers.**

En Maple, les opérations sur des nombres entiers ou rationnels sont *exactes*.

Exemple 1. Un calcul typique d'entier est celui de factorielle 100.

```
100!;
```

$$933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000$$

De nombreuses fonctions s'appliquent aux entiers.

Exemple 2. Fermat avait conjecturé que tous les nombres de la forme $2^{2^n}+1$ étaient premiers. Voici le premier exemple qui invalide sa conjecture :

```
ifactor(2^(2^5)+1);
```

$$(641)(6700417)$$

Du point de vue de la simplification, tous les entiers sont représentés en base dix (ou deux, ou une puissance de deux), ce qui constitue une forme normale. L'égalité d'entiers est donc facile à tester (en Maple, le test d'égalité syntaxique se fait en temps constant, indépendamment de la taille des objets). Toute opération sur des entiers est immédiatement effectuée ; par exemple, 2^2 n'est pas représentable en Maple, il est immédiatement transformé en 4. Cela signifie aussi qu'un nombre factorisé ne peut pas être représenté comme un entier, puisqu'alors il serait immédiatement développé. Dans l'exemple précédent, le résultat est en réalité un produit de fonctions.

Tab. 1. Principaux simplificateurs

Classe d'expressions	Fonction
entiers	simplification automatique
rationnels	simplification automatique
flottants	`evalf`
booléens	`evalb`
résidus mod p	`mod`
nombres p-adiques	`padic[evalp]`
matrices	`evalm`
fractions rationnelles	`normal`
développements limités	`series`
nombres algébriques	`evala`
racines carrées	`rationalize`
nombres complexes	`evalc`
fonction f	`simplify(...,f)`

1.2. **Rationnels.** La propriété de forme normale s'étend aux nombres rationnels. Non seulement les additions, multiplications et quotients sont immédiatement exécutés, mais en plus les fractions rationnelles sont toutes réduites.

Exemple 3. Dans cet exemple, les factorielles sont d'abord évaluées, puis le rationnel obtenu est simplifié :

```
99!/100!-1/50;
```

$$-\frac{1}{100}$$

1.3. **Flottants.** Les règles de simplification automatique sont moins systématiques pour les nombres approchés numériquement, appelés aussi nombres en virgule flottante, ou plus simplement *flottants*. Lorsqu'ils interviennent dans une somme, un produit ou un quotient faisant intervenir par ailleurs des rationnels, ils sont contagieux, c'est-à-dire que toute l'expression devient un nombre flottant.

Exemple 4.

```
72/53-5/3*2.7;
```

$$-3.141509435$$

Pour les autres expressions, la fonction de base pour ces calculs est `evalf` qui évalue numériquement une expression (tous les nombres sont transformés en flottants). Un argument optionnel permet de préciser le nombre de chiffres significatifs utilisés lors du calcul.

Exemple 5. Voici π avec 50 chiffres significatifs

```
evalf(Pi,50);
```

$$3.1415926535897932384626433832795028841971693993751$$

La précision peut également être réglée par la variable globale `Digits`, qui vaut 10 par défaut.

Les flottants en Maple sont liés à leur précision : ainsi la valeur précédente est différente syntaxiquement de la valeur de π calculée avec dix chiffres significatifs. Compte tenu de cette restriction, les flottants renvoyés par `evalf` sont sous forme normale.

1.4. **Booléens.** Les expressions booléennes forment aussi une classe élémentaire. Les deux formes normales sont `true` et `false`. Les autres expressions s'y réduisent par la commande `evalb`.

Exemple 6.

```
a:=0:b:=2:c:=3:
evalb(a=1 or (b=2 and c=3));
```

true

1.5. **Classes issues de l'arithmétique.** Les autres constantes formant une classe élémentaire munie d'une forme normale sont les résidus modulo p, avec pour fonction de réduction `mod`, et les nombres p-adiques, mis sous forme normale par la fonction `padic[evalp]`.

TAB. 2. Récritures de polynômes

Polynôme `p`	$zx^2 + x^2 - (x^2 + y^2)(ax - 2by) + zy^2 + y^2$
`collect(p,x)`	$(z + 1 + 2by)x^2 - y^2ax + 2y^3b + zy^2 + y^2 - x^3a$
`collect(p,[x,y])`	$(z + 1 + 2by)x^2 - y^2ax + 2y^3b + (z + 1)y^2 - x^3a$
`collect(p,[x,y], distributed)`	$(z + 1)x^2 + 2byx^2 - y^2ax + 2y^3b + (z + 1)y^2 - x^3a$
`expand(p)`	$zx^2 + x^2 - x^3a + 2x^2by - y^2ax + 2y^3b + zy^2 + y^2$
`factor(p)`	$(x^2 + y^2)(-ax + z + 1 + 2by)$

2. Classes à forme normale

À partir de constantes bien définies, des classes d'objets symboliques faisant intervenir des variables et admettant une forme normale peuvent être construites. Les plus importantes sont les matrices, les polynômes et fractions rationnelles, les développements limités et les nombres algébriques. Pour chacune de ces classes, nous indiquons les principales fonctions de récriture.

2.1. **Matrices.** La forme normale d'une matrice est obtenue lorsque tous ses coefficients sont eux-mêmes sous forme normale.

Il y a plusieurs types de matrices en Maple. Pour le type `Matrix`, qui est à préférer, les simplifications sont automatiques. Il faut cependant penser à utiliser `.` et non `*` pour la multiplication, seul le premier garantissant la non-commutativité.

Exemple 7. `a:=Matrix([[1,2,3],[2,4,8],[3,9,27]]);`

$$a := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix}$$

`(a^2+ 1) . a^(-1) ;`

$$\begin{bmatrix} -5 & 13/2 & 7/3 \\ 7 & 1 & 25/3 \\ 2 & 19/2 & 27 \end{bmatrix}$$

2.2. **Polynômes et fractions rationnelles.** Les calculs sur les polynômes et les fractions rationnelles à une ou plusieurs indéterminées sont les opérations de base d'un système de calcul formel. Contrairement aux classes présentées jusqu'ici, il n'y a pas *une* bonne représentation des polynômes. Les fonctions permettant de *récrire* un polynôme sous diverses formes sont résumées au tableau 2. Le tableau 3 résume celles concernant les fractions rationnelles.

Le dernier argument de `collect` est une procédure qui est appliquée à chacun des coefficients, ce qui peut être très utile. La commande `convert(.,parfrac,.)` effectue la décomposition en éléments simples.

2.3. **Développements limités.** Comme les matrices et les flottants, les développements limités ont une forme normale, mais celle-ci n'est pas produite automatiquement. La commande de réduction est `series`. Comme pour `evalf`, l'ordre des développements est spécifié soit en donnant un argument supplémentaire à `series`, soit en modifiant la variable globale *Order*, qui vaut 6 par défaut.

Exemple 8.

TAB. 3. Récritures de fractions rationnelles

Fraction `f`	Opération	Résultat
$\frac{x^3+3x^2+2x+yx^2+3yx+2y}{x^3+yx+2x^2+2y}$	`normal(f)`	$\frac{x^2+yx+x+y}{x^2+y}$
	`factor(f)`	$\frac{(x+1)(x+y)}{x^2+y}$
$\frac{x^3+3x^2+2x+yx^2+3yx+2y}{x^3+x+2x^2+2}$	`collect(f,y)`	$\frac{(x^2+3x+2)y}{x^3+x+2x^2+2}+\frac{x^3+3x^2+2x}{x^3+x+2x^2+2}$
	`collect(f,y,normal)`	$\frac{(x+1)y}{x^2+1}+\frac{(x+1)x}{x^2+1}$
$\frac{x^2+3yx+2}{x^2+1}$	`expand(f)`	$\frac{x^2}{x^2+1}+\frac{3yx}{x^2+1}+\frac{2}{x^2+1}$
$\frac{3x^2y+3xy^2+1}{x^2-4y^2}$	`convert(f,parfrac,x)`	$3y+\frac{1+18y^3}{4y(x-2y)}-\frac{1+6y^3}{4y(x+2y)}$

```
series(x/(2+x-x^2+O(x^3)),x);
```

$$\frac{1}{2}x-\frac{1}{4}x^2+\frac{3}{8}x^3+O(x^4)$$

```
s:=series(exp(sin(log(1+x))),x);
```

$$s := 1+x-\frac{1}{6}x^3+\frac{1}{12}x^4-\frac{1}{30}x^5+O(x^6)$$

```
series(s^2+x^3,x,4);
```

$$1+2x+x^2+\frac{2}{3}x^3+O(x^4)$$

Il est important de noter que si les coefficients des développements limités sont dans une classe d'expressions n'admettant pas de forme normale, alors les résultats renvoyés par `series` peuvent être faux.

Exemple 9. Une des façons les plus simples de déguiser 0 consiste à l'écrire $\exp(1)\exp(-1)-1$. Ceci conduit à une erreur de `series` :

```
f:=1/(z-z*exp(z)*exp(-1/z));
```

$$f := \frac{1}{z-ze^ze^{(-\frac{1}{z})}}$$

```
series(f,z=1,2);
```

$$\frac{1}{1-ee^{-1}}+\frac{1-3ee^{-1}}{(ee^{-1}-1)(1-ee^{-1})}(z-1)+O((z-1)^2)$$

Le premier terme est en réalité infini, `series` devrait repérer un pôle simple et produire $(1-z)^{-1}/2+O(1)$, ce que l'on obtient en appliquant `series` à `combine(f,exp)` (cf. tab. 5 p. 9).

2.4. **Nombres algébriques.** Un nombre algébrique est défini comme racine d'un polynôme. Lorsque le degré du polynôme est plus grand que 4, il n'est pas possible de le résoudre explicitement en général. Cependant, de nombreux calculs sur ses racines peuvent être menés à bien sans autre information que le polynôme lui-même.

Les nombres algébriques sont représentés en Maple par l'opérateur `RootOf` qui prend en argument le polynôme. Les fractions rationnelles en un nombre algébrique admettent une forme normale, calculée par `evala`.

Exemple 10.

```
alias(alpha=RootOf(x^7+3*x^2+1,x)):
alpha^3/(alpha^8+3*alpha^2+1);
```

$$\frac{\alpha^3}{\alpha^8+3\alpha^2+1}$$

```
evala(%);
```

$$-\frac{1}{5}\alpha^2-\frac{34}{5}-\frac{11}{5}\alpha^6-\frac{11}{5}\alpha^5+\frac{4}{5}\alpha^4+\frac{4}{5}\alpha^3-\frac{34}{5}\alpha$$

Il faut noter que l'expression `RootOf(x^7+3*x^2+1,x)`, que nous avons fait afficher α pour une meilleure lisibilité à l'aide de la commande `alias`, représente l'une *quelconque* des sept racines du polynôme x^7+3x^2+1. Par contre, l'identité prouvée par `evala` n'est vraie que si les différentes occurrences du symbole α sont remplacées par la *même* racine.

2.5. **Racines carrées.** Pour simplifier des fractions rationnelles dont le dénominateur comprend des racines carrées, la méthode classique de multiplication par l'expression conjuguée est réalisée par la commande `rationalize`.

```
rationalize(1/(1+sqrt(2)+sqrt(3)));
```

$$-\frac{1}{4}(-1-\sqrt{3}+\sqrt{2})(-1+\sqrt{3})$$

Pour obtenir une forme normale, il faut développer le résultat donné par la commande `rationalize` ; les facteurs obtenus au numérateur dépendent en effet de l'ordre d'élimination des racines carrées.

```
expand(%);
```

$$\frac{1}{2}+\frac{1}{4}\sqrt{2}-\frac{1}{4}\sqrt{2}\sqrt{3}$$

Nous aurions pu obtenir le même résultat (mais plus laborieusement) à l'aide de `RootOf` et `evala`, en substituant `RootOf(x^2=n)` à `sqrt(n)`, puis en appliquant `evala`, et en effectuant la substitution inverse.

La commande `rationalize` accepte également des expressions contenant des racines imbriquées ou des variables :

```
rationalize(1/(sqrt(x-sqrt(y))+sqrt(z+t)));
```

$$-\frac{\left(-\sqrt{z+t}+\sqrt{x-\sqrt{y}}\right)\left(z+t-x-\sqrt{y}\right)}{z^2+2tz-2xz+t^2-2tx+x^2-y}$$

3. Expressions complexes et simplification

Les classes d'expressions présentées jusqu'ici partagent la propriété d'avoir une procédure de décision pour la nullité. C'est-à-dire que pour toutes ces classes un programme peut déterminer si une expression donnée est nulle ou non. Dans de nombreux cas, cette décision se fait par réduction à la forme normale : l'expression est nulle si et seulement si sa forme normale est le symbole 0.

Malheureusement, toutes les classes d'expressions n'admettent pas une forme normale. Pire encore, pour certaines classes il est impossible de prouver la nullité d'une expression en temps fini. Un exemple d'une telle classe est fourni par les expressions composées à partir des rationnels, des nombres π et $\log 2$ et d'une variable, par utilisation répétée de l'addition, de la soustraction, du produit, de l'exponentielle et du sinus. Bien sûr, une utilisation répétée de `evalf` en augmentant la précision permet souvent de savoir si une expression particulière est nulle ou non ; mais Richardson a montré qu'il est impossible d'écrire un programme prenant en argument une expression de cette classe et donnant au bout d'un temps fini le résultat vrai si celle-ci est nulle, et faux sinon.

C'est dans ces classes que se pose avec le plus d'acuité le problème de la simplification. Sans forme normale, les systèmes ne peuvent que donner un certain nombre de fonctions de récriture avec lesquelles l'utilisateur doit jongler pour parvenir à un résultat. Pour y voir plus clair dans cette jungle, il faut là encore distinguer plusieurs sous-classes, savoir quelles fonctions s'appliquent et quelles transformations sont effectuées.

3.1. **Constantes.** Comme dit précédemment, les calculs se font avec des nombres entiers ou rationnels *exacts* et avec des constantes mathématiques *vraies* (qui ne sont pas des représentations flottantes).

Les constantes les plus simples sont les rationnels, le nombre π noté `Pi`, la base e des logarithmes népériens notée `E`, le nombre imaginaire i noté `I` et la constante d'Euler γ notée `gamma`.

Ces constantes sont relativement bien connues du système. Une exception est la constante `E`, peu utilisée par Maple, à laquelle il faut préférer `exp(1)`. Pour la classe simple des polynômes en π et e, aucun algorithme de décision n'est connu : à ce jour on ignore s'il existe un tel polynôme non trivial qui vaille zéro.

En utilisation interactive, une bonne façon de traiter ces constantes dans des simplifications compliquées est de les remplacer toutes sauf i par des variables et d'utiliser les procédures de forme normale des fractions rationnelles. Ceci revient à faire l'hypothèse que toutes ces constantes sont algébriquement indépendantes. Cette remarque se généralise à des constantes plus complexes comme $\ln 2$, $\exp(\pi + \log 3)$,... mais il faut alors être sûr que celles-ci ne sont pas trivialement dépendantes.

3.2. **Nombres complexes.** En Maple, on note `I` le nombre imaginaire i.

La fonction de base pour les calculs sur les nombres complexes est `evalc`. Elle met une expression sous la forme $a+ib$, où a et b sont réels. Comme la nullité n'est pas en général décidable, il en va de même de la réalité. Cependant, dès que a et b sont dans des classes à forme normale, `evalc` fournit une forme normale pour les complexes associés.

Exemple 11. On peut par exemple calculer $\sqrt{i}$:

```
(I)^(1/2);
```

$$(-1)^{1/4}$$

```
evalc(%);
```

$$\frac{1}{2}\sqrt{2} + \frac{1}{2}I\sqrt{2}$$

Le résultat de ce calcul pose le problème de la détermination des racines. L'imaginaire i a deux racines alors que `evalc` n'en donne qu'une. Dans le cas d'expressions plus compliquées, les choix multiples de racines carrées ou cubiques peuvent rendre la reconnaissance de 0 difficile, surtout si ces choix doivent être faits de manière cohérente dans l'expression.

Le même type de problème se pose avec toutes les fonctions multiformes, comme le logarithme ou les fonctions hypergéométriques. Le système fournit alors peu d'assistance pour les simplifications.

Par ailleurs, dans un calcul avec des expressions complexes, `evalc` suppose que les variables qui interviennent sont réelles. Tout ceci entraîne qu'il faut être très prudent avec la manipulation de nombres complexes.

Les autres commandes principales sont `Re`, `Im`, `abs` et `argument` donnant respectivement la partie réelle, la partie imaginaire, le module et l'argument.

```
z:=a+I*b: Re(z),Im(z),abs(z),argument(z);
```

$$\Re(a) - \Im(b), \Im(a) + \Re(b), |a + Ib|, \text{argument}(a + Ib)$$

Un appel à `evalc` simplifie en supposant a et b réels :

```
evalc(Re(z)),evalc(Im(z)),evalc(abs(z)),evalc(argument(z));
```

$$a, b, \sqrt{a^2 + b^2}, \arctan(b, a)$$

3.3. **Fonctions.** La plupart des fonctions mathématiques usuelles se retrouvent en Maple, en particulier les fonctions trigonométriques, le logarithme et l'exponentielle. La simplification de telles fonctions est cruciale. Le tableau 5 p. 9 décrit les commandes de base réalisant ces simplifications.

Tout ce qui concerne le classique tableau de variation de la fonction (calcul des dérivées, des asymptotes, des extremums, recherche des zéros et tracé de la courbe) peut être facilement réalisé. Les principales opérations Maple qui s'appliquent à une fonction sont résumées au tableau 6 p. 9.

3.4. **Équations.** Un principe important du calcul formel est la manipulation d'objets définis par des équations, sans passer par la résolution de celles-ci.

Ainsi, une fonction définie par une équation différentielle linéaire et des conditions initiales est parfaitement précisée. L'ensemble des solutions d'équations différentielles linéaires est clos par addition et produit (entre autres) et forme ainsi une importante classe où l'on peut décider de la nullité. En revanche, si l'on résout une telle équation, la solution, privée de son équation de définition, tombe dans une classe plus grande où bien peu est décidable. Cependant, dans certains cas, surtout en utilisation interactive, il est utile de chercher une solution explicite, par exemple pour passer à une application numérique. Les principales fonctions de résolution sont résumées au tableau 4.

Tab. 4. Résolution d'équations

Commande	Usage
`fsolve`	solutions flottantes
`isolve`	solutions entières
`msolve`	solutions modulaires
`linsolve`	solutions d'équations linéaires
`dsolve`	solutions d'équations différentielles
`rsolve`	solutions de récurrences
`solve`	résolveur symbolique général

3.5. **Formes inertes en Maple.** Dans l'exemple 10 p. 5, nous avons représenté les racines du polynôme $x^7 + 3x^2 + 1$ à l'aide de `RootOf`. Cette fonction ne fait aucun calcul ; elle sert uniquement à représenter l'une des solutions d'une équation. De telles fonctions qui ne font rien sont appelées des fonctions *inertes*. Leur rôle est d'être reconnues par les fonctions du système qui effectuent des calculs.

`RootOf` n'est pas la seule fonction inerte de Maple. La fonction `DESol` représente les solutions d'une équation différentielle. La fonction `Int` représente une intégrale

TAB. 5. Simplifications des fonctions élémentaires

Commande	Résultat
`expand(···)`	$\sin(a+b) \mapsto \sin(a)\cos(b)+\cos(a)\sin(b)$
	$\cos(a+b) \mapsto \cos(a)\cos(b)-\sin(a)\sin(b)$
	idem pour les fonctions hyperboliques
	$e^{a+b} \mapsto e^a e^b$
`combine(···,trig)`	$\cos(a)\cos(b) \mapsto \cos(a-b)/2+\cos(a+b)/2$
	$\cos(a)\sin(b) \mapsto \sin(a+b)/2-\sin(a-b)/2$
	$\sin(a)\sin(b) \mapsto \cos(a-b)/2-\cos(a+b)/2$
	idem pour les fonctions hyperboliques
`combine(···,exp)`	$e^a e^b \mapsto e^{a+b}$
	$(e^a)^b \mapsto e^{ab}$
	$e^{a+n\ln b} \mapsto b^n e^a$ où n est entier
`combine(···,ln)`	$n\ln b \mapsto \ln(b^n)$ où n est entier
	$\ln a+\ln b \mapsto \ln(ab)$
`simplify(···,trig)`	$\sin(x)^2+\cos(x)^2 \mapsto 1$
	$\cosh(x)^2-\sinh(x)^2 \mapsto 1$
`simplify(···,exp)`	$e^{a\ln b} \mapsto b^a$
`simplify(···,ln)`	$\ln(b^a) \mapsto a\ln b$
ou `expand(···)`	$\ln(ab) \mapsto \ln a+\ln b$
`simplify(···,power)`	$x^a x^b \mapsto x^{a+b}$
`simplify(···,power,symbolic)`	$(a^b)^c \mapsto a^{bc}$
ou `combine(···,power)`	$\sqrt{a^2} \mapsto a$
`convert(···,exp)`	$\cos(x) \mapsto (e^{ix}+e^{-ix})/2$
	$\cosh(x) \mapsto (e^x+e^{-x})/2$
`convert(···,trig)`	$e^{ix} \mapsto \cos(x)+i\sin(x)$
	$e^x \mapsto \cosh(x)+\sinh(x)$
`convert(···,ln)`	$\arccos(x) \mapsto -i\ln(x+i\sqrt{1-x^2})$
	$\operatorname{arctanh}(x) \mapsto (\ln(1+x)-\ln(1-x))/2$

TAB. 6. Principales opérations sur les fonctions

Expression	Résultat
`f`	la fonction elle-même
`diff(f,x)`	dérivée par rapport à x
`int(f,x)`	primitive par rapport à x
`eval(subs(x=a,f))`	$f(a)$
`limit(f,x=a)`	limite de $f(x)$ lorsque $x \to a$
`series(f,x=a)`	développement limité en $x=a$
`asympt(f,x)`	développement asymptotique lorsque $x \to \infty$
`plot(f,x=a..b)`	la courbe $y=f(x)$ pour $x \in [a,b]$

TAB. 7. Calcul avec des formes inertes

Forme inerte	Fonctions qui s'y appliquent
`Int`	`diff`, `evalf`, `series`
`Sum`	`diff`, `evalf`
`Product`	`diff`, `evalf`, `mod`
`RootOf`	`evala`, `sum`, `factor`, `allvalues`, `testeq`, `solve`
	`evalf`, `product`, `diff`, `evalc`, `series`
`DESol`	`diff`, `series`
`Diff`	`diff`, `D`, `liesymm`, `expand`
`Limit`	`evalf`
`Re`, `Im`	`evalc`
`Eigenvals`	`Svd`, `evalf`

que le système ne cherche pas à calculer (contrairement à `int`). Ainsi, la différence entre

`evalf(Int(f,x=a..b))` et `evalf(int(f,x=a..b))`

est que dans le premier cas, la routine d'évaluation numérique d'intégrale est immédiatement appelée, alors que dans le second cas, le système cherche d'abord une forme symbolique. S'il en trouve une, par exemple lorsque $f = \cos x$, elle est employée ; sinon la routine d'évaluation numérique est appelée. Par exemple lorsque $f = \cos(\sin x)$, après avoir perdu du temps à chercher en vain une solution symbolique, le système réalise l'évaluation numérique. La fonction `Sum` joue le même rôle pour les sommes.

Lorsqu'une expression contient des formes inertes, la procédure `value` les rend actives afin de calculer la valeur de l'expression. Pour `RootOf`, la commande `allvalues` joue partiellement ce rôle.

Les principales formes inertes peuvent être récrites par `combine`, `expand`, `simplify` et certaines commandes du *package* `student`. Le tableau 7 montre les autres fonctions qui prennent en compte les formes inertes.

La forme inerte la mieux connue du système est `RootOf`. Son argument n'est pas forcément un polynôme ; les commandes `evalf` et `series` savent aussi traiter des équations plus générales. Voici par exemple une façon de trouver numériquement une racine de l'équation $z + \cos z = 2$

```
evalf(RootOf(z+cos(z)=2));
```

$$2.988268926$$

et voici comment obtenir un développement limité à l'origine de la fonction $y(x)$ définie implicitement par $y \exp(x) = y^5 + \ln(1 + x)$:

```
series(RootOf(y*exp(x)=y^5+ln(1+x),y),x);
```

$$x - \frac{3}{2}x^2 + \frac{4}{3}x^3 - x^4 + \frac{209}{120}x^5 + O(x^6)$$

4. HYPOTHÈSES SUR LES VARIABLES

Les variables non affectées posent problème lors des calculs. Par exemple, que vaut $\sqrt{x^2}$? Si x est réel positif, on voudrait obtenir x, si x est réel négatif, on voudrait obtenir $-x$ et si x est un nombre complexe quelconque, on doit choisir

parmi deux racines complexes opposées. Ce problème du type de la variable se pose dans bien d'autres cas.

Dans certains cas, le problème peut être réglé par l'emploi de l'option `symbolic` de la fonction `simplify` qui permet de réaliser les simplifications sans se poser de question. Dans le cas général, la bonne solution consiste à utiliser la fonction `assume` qui permet de préciser les propriétés d'une variable.

La fonction `about` indique les hypothèses faites sur une variable. Après l'appel de `assume`, les variables sont automatiquement renommées et un tilde apparaît lors de leur impression. Mais on continue à les utiliser normalement en entrée.

Exemple 12.
```
assume(R1>0); assume(R2>0);
about(R1);
    Originally R1, renamed R1~:
      is assumed to be: RealRange(Open(0),infinity)
```

Avec ces hypothèses sur R_1 et R_2 des simplifications sont réalisées automatiquement qui ne l'auraient pas été sinon :

```
A:=normal(R1^2*(arccos(c1)-s1*c1)+R2^2*(arccos(c2)-s2*c2));
```

$$A := \arccos\left(\frac{1}{2}\frac{2R1^{\sim 2} - R2^{\sim 2}}{R1^{\sim 2}}\right) R1^{\sim 2} - \frac{1}{2} R2^{\sim}\sqrt{4R1^{\sim 2} - R2^{\sim 2}} + R2^{\sim 2}\arccos\left(\frac{1}{2}\frac{R2^{\sim}}{R1^{\sim}}\right)$$

Le seul moyen de supprimer les hypothèses faites sur une variable est de lui donner une valeur (qui peut être son nom). Par ailleurs les nouvelles hypothèses n'ont pas d'effet rétroactif et il faut éventuellement refaire des calculs.

Exemple 13. La variable `e` garde sa valeur obtenue pour $a > 0$ même après avoir spécifié que a est négatif.

```
assume(a>0); e:=1+sqrt(a^2);
```

$$e := 1 + a^{\sim}$$

```
assume(a<0); e;
```

$$1 + a^{\sim}$$

```
e:=1+sqrt(a^2);
```

$$e := 1 - a^{\sim}$$

Pour résumer, voici les principales fonctions manipulant des hypothèses :
- `assume` déclare une hypothèse ;
- `additionally` rajoute une hypothèse à une variable ;
- `isgiven` teste si une hypothèse a été déclarée ;
- `is` teste si une hypothèse se déduit de celles qui ont été déclarées ;
- `about` liste les hypothèses faites sur une variable.

Dans le tableau 8, nous indiquons les principales propriétés utilisables et une partie des fonctions qui en tiennent compte. Ce tableau est très incomplet, puisque les fonctions qui utilisent `signum` (fonction signe pour les expressions réelles ou complexes) par exemple tiennent également compte de ces propriétés.

TAB. 8. Principales propriétés reconnues par Maple

Déclaration	Fonctions l'exploitant
`assume(x<>0)`	`signum`
`assume(x>=0)`	`signum`, `abs`, `csgn`, `int`
`assume(x<=0)`	`signum`, `abs`, `csgn`, `int`
`assume(x>0)`	`signum`, `abs`, `csgn`, `int`
`assume(x<0)`	`signum`, `abs`, `csgn`, `int`
`assume(x,integer)`	`floor`, `frac`, `round`, `trunc`
`assume(x,real)`	`signum`, `abs`, `Re`, `Im`

5. OBJETS COMPOSÉS

Maple gère des suites d'expressions, des ensembles et des listes. Le tableau 9 décrit les caractéristiques et différences de ces objets. Un dernier type d'objet composé est la table, mais les règles d'évaluation des tables rendent leur utilisation un peu délicate et sort de ce petit guide.

TAB. 9. Listes, ensembles et suites d'expressions

Objet	Caractéristiques
liste	type : `list` syntaxe : `[a,b,c]` ordonné : `[a,b,c]` $\neq$ `[b,a,c]` éléments répétés : `[a,a,c]` $\mapsto$ `[a,a,c]` plusieurs niveaux : `l :=[b,c] ;  [a,l,d]` $\mapsto$ `[a,[b,c],d]` liste vide : `[]`
ensemble	type : `set` syntaxe : {a,b,c} non ordonné : {a,b,c} = {b,a,c} pas d'éléments répétés : {a,a,c}$\mapsto${a,c} plusieurs niveaux : `l :={b,c} ;  {a,l,d}` $\mapsto$ {a,{b,c},d} ensemble vide : {}
suite d'expressions	type : `exprseq` syntaxe : `a,b,c` ordonné : `a,b,c` $\neq$ `b,a,c` éléments répétés : `a,a,c` $\mapsto$ `a,a,c` un seul niveau : `l :=b,c ;  a,l,d` $\mapsto$ `a,b,c,d` suite vide : `NULL`

6. OPÉRATEURS FONCTIONNELS

Une technique courante pour donner une valeur à une expression consiste à utiliser la fonction `eval`. Une autre façon de réaliser cette opération est d'utiliser un *opérateur fonctionnel*.

En Maple la notation flèche `->` crée une fonction de zéro ou plusieurs arguments.

Exemple 14. Voici comment définir une fonction f qui à (x, y) associe l'expression $x^2 + y^2 - 1$

```
f:=(x,y)->x^2+y^2-1;
```

$$f := (x, y) \to x^2 + y^2 - 1$$

```
f(1,a);
```

$$a^2$$

La fonction f s'évalue alors comme n'importe quelle fonction Maple.

Inversement, à partir d'une expression algébrique, on peut obtenir une fonction. Pour cela on utilise la fonction `unapply`, équivalent de l'abstraction du λ-calcul.

En Maple on peut même composer ou dériver les opérateurs fonctionnels à l'aide de l'opérateur de composition `@@` et de l'opérateur de dérivation `D`. L'opérateur `@@@@` est l'opération de composition itérée. Ainsi, `f@@f@@f` et `f@@@@3` représentent tous deux la troisième itérée de f.

Exemple 15.

```
f:=x->x^x;
```

$$f := x \to x^x$$

```
(f@@f)(x);
```

$$(x^x)^{(x^x)}$$

```
(D@@@@2)(f@@f)(x);
```

$$\left((x^x)^{(x^x)}(\ln(x^x)+1)^2 + \frac{(x^x)^{(x^x)}}{x^x}\right)(x^x)^2(\ln(x)+1)^2 + (x^x)^{(x^x)}(\ln(x^x)+1)\left(x^x(\ln(x)+1)^2 + \frac{x^x}{x}\right)$$

Malgré toutes ces possibilités offertes par les opérateurs fonctionnels, nous conseillons de travailler plutôt sur des expressions pour manipuler des fonctions !— avec `eval` ou `subs` pour donner des valeurs aux paramètres. Bien que l'évaluation soit un peu moins naturelle pour un mathématicien, les expressions se prêtent à des opérations plus nombreuses. Par exemple, Maple sait calculer une primitive d'une expression, mais ne peut le faire pour un opérateur fonctionnel. Il en va de même pour de nombreuses commandes dont l'argument doit être une expression.

7. Exercices

Ces exercices visent essentiellement à faire exécuter des calculs simples en apprenant progressivement à se débrouiller à l'aide de la documentation en ligne. Pour chaque exercice, sont indiqués entre crochets les items de la documentation en ligne à consulter.

(1) Calculer la valeur numérique de $e^{\pi\sqrt{163}} - 262537412640768744$ avec successivement 10, 20, 30, 40 et 50 chiffres significatifs. Commentaires. [`evalf`]

(2) Montrer que $(z+1)(z+j)(z+j^2) = (1+z)(1+jz)(1+j^2z)$ où j est une racine cubique de l'unité. Pour cela on définira j comme un nombre algébrique. [`RootOf`, `evala`, `alias`]

(3) Retrouver les formules développées de $\sin(3x)$, $\cos(3x)$ et $\mathrm{ch}(5x)$. [`expand`, `inifcns`]